

OPENMP IMPLEMENTATION

Summary

GitHub 4

Description problem..... 4

Description solution 5

Development..... 7

Results 9

Conclusions 10

List of images

Figura 1 - Knapsack problem	4
Figura 2 - Knapsack serial problem	5
Figura 3 - Knapsack parallel problem.	6
Figura 4 - Modifications on the code.	7
Figura 5 - Using the qlogin command to connect with the one computer.	7
Figura 6 - File batch to execute all files (Serial/Parallel (Thread2/4/8).	8
Figura 7 - Results	9

GitHub



<https://github.com/tankintat/highperformanceudl>

Description problem

0-1 KNAPSACK PROBLEM

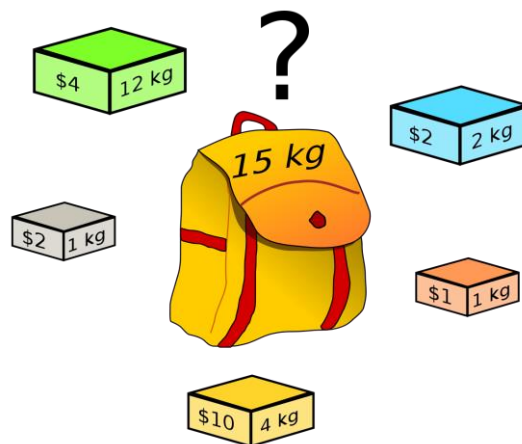


Figura 1 - Knapsack problem

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays $val[0..n-1]$ and $wt[0..n-1]$ which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of $val[]$ such that sum of the weights of this subset is smaller than or equal to W . You cannot break an item, either pick the complete item, or don't pick it (0-1 property).

Description solution

The problem has two ways to resolve, Dynamic Programming approach or Branch and Bounce approach to organize the idea to paralelize the algorithm in order to increase the speed of execution. The serial algorithm execute runs linearly that means try to found the better way searching all ways to find the maximum weight with the better value.

The serial solution has 2 loops, one to control the quantity of inputs, and the inside control each weight increasing one by one up to the maximum weight. This makes it possible to test all combinations.

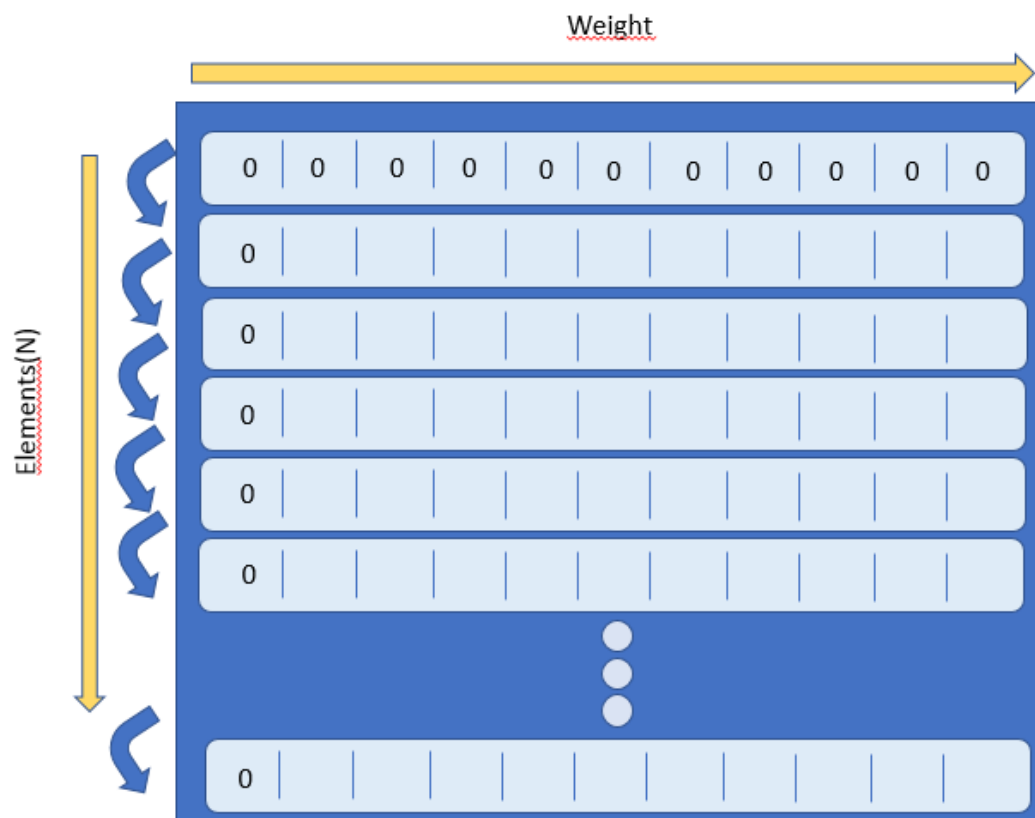


Figura 2 - Knapsack serial problem

On the inside loop to execute in parallel all weights the same time and the barrier to wait and restart to next one again. We can put on the external loop because on the algorithm need the last row when overcome the sack weight in this case its necessary to get the last sum weight.

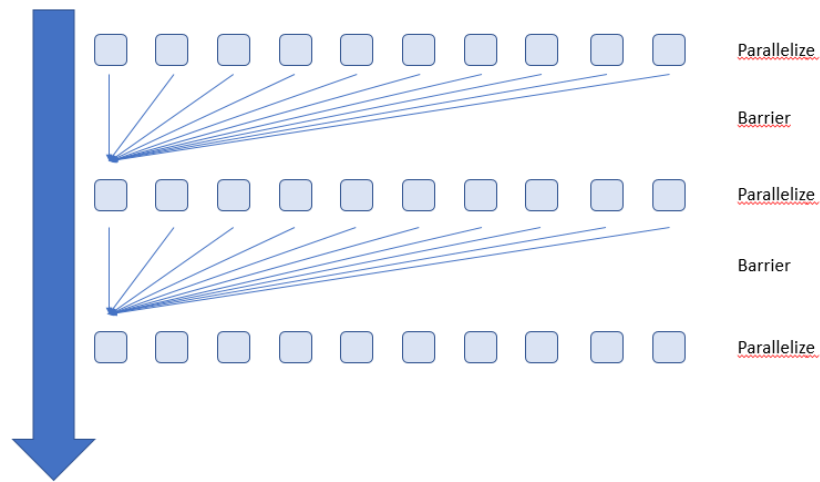
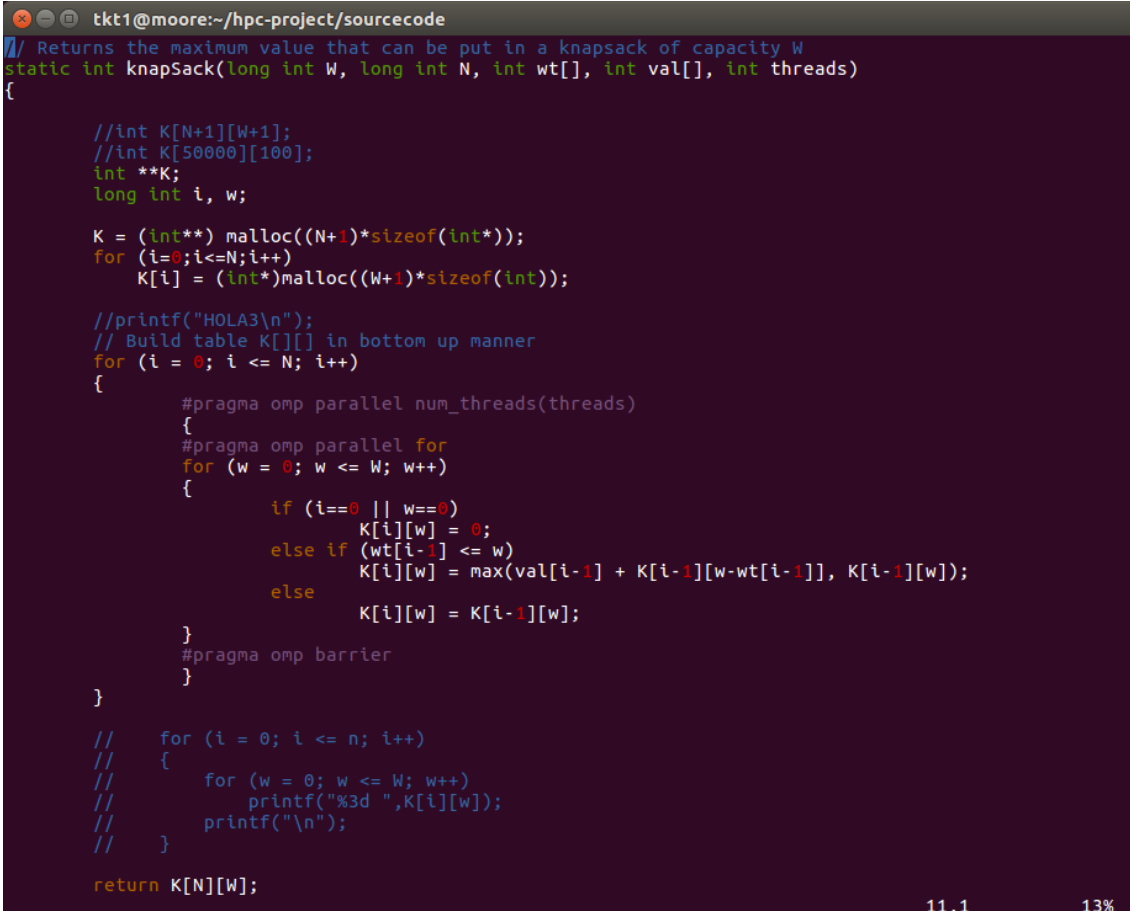


Figura 3 - Knapsack parallel problem.

Development

For the development it was necessary the server moore of the University used to the tests. The first was connected of the moore server and put the folder base (hpc-project) to start.



```
tk1@moore:~/hpc-project/sourcecode
// Returns the maximum value that can be put in a knapsack of capacity W
static int knapSack(long int W, long int N, int wt[], int val[], int threads)
{
    //int K[N+1][W+1];
    //int K[50000][100];
    int **K;
    long int i, w;

    K = (int**) malloc((N+1)*sizeof(int*));
    for (i=0;i<=N;i++)
        K[i] = (int*)malloc((W+1)*sizeof(int));

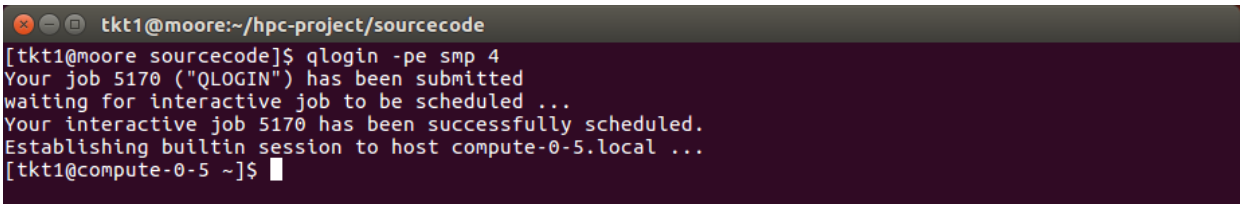
    //printf("HOLA3\n");
    // Build table K[][] in bottom up manner
    for (i = 0; i <= N; i++)
    {
        #pragma omp parallel num_threads(threads)
        {
            #pragma omp parallel for
            for (w = 0; w <= W; w++)
            {
                if (i==0 || w==0)
                    K[i][w] = 0;
                else if (wt[i-1] <= w)
                    K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
                else
                    K[i][w] = K[i-1][w];
            }
            #pragma omp barrier
        }
    }

    // for (i = 0; i <= n; i++)
    // {
    //     for (w = 0; w <= W; w++)
    //         printf("%3d ",K[i][w]);
    //     printf("\n");
    // }

    return K[N][W];
}
```

Figura 4 - Modifications on the code.

Using the qlogin command to execute the serial and parallel programs.



```
tk1@moore:~/hpc-project/sourcecode
[tk1@moore sourcecode]$ qlogin -pe smp 4
Your job 5170 ("QLOGIN") has been submitted
waiting for interactive job to be scheduled ...
Your interactive job 5170 has been successfully scheduled.
Establishing builtin session to host compute-0-5.local ...
[tk1@compute-0-5 ~]$
```

Figura 5 - Using the qlogin command to connect with the one computer.

And the end execute a file batch to execute all programs serials and parallels to compare the outputs. Compile the file before to execute the running_results.sh.

```
tk1@moore:~/hpc-project/sourcecode
#!/bin/bash

for file in $HOME/hpc-project/testbed/*
do
    echo "Running $file"
    echo "===== " >> $HOME/hpc-project/sourcecode/results
    echo "File $file: " >> $HOME/hpc-project/sourcecode/results
    echo "Serial"
    echo "Serial: " $(./knapsackDYN $file) >> $HOME/hpc-project/sourcecode/results
    echo "Omp2"
    echo "OMP 2: " $(./knapsackDYNOMP $file 2) >> $HOME/hpc-project/sourcecode/results
    echo "Omp4"
    echo "OMP 4: " $(./knapsackDYNOMP $file 4) >> $HOME/hpc-project/sourcecode/results
    echo "Omp8"
    echo "OMP 8: " $(./knapsackDYNOMP $file 8) >> $HOME/hpc-project/sourcecode/results
    echo "===== " >> $HOME/hpc-project/sourcecode/results
done
```

Figura 6 - File batch to execute all files (Serial/Parallel (Thread2/4/8)).

Results

The complete results you can see opening this link:

[https://github.com/tankintat/highperformanceudl/blob/master/OpenMP Implementations/results](https://github.com/tankintat/highperformanceudl/blob/master/OpenMP%20Implementations/results) or the file "result" package together with this file.

```
tk1@moore:~/hpc-project/sourcecode
File /home/tkt1/hpc-project/testbed/test_500_1000:
Serial: 1000:500:1722:0.008256:0.008674
OMP 2: 1000:500:1722:0.011011:0.011219
OMP 4: 1000:500:1722:0.008174:0.008402
OMP 8: 1000:500:1722:0.047525:0.047625
=====
File /home/tkt1/hpc-project/testbed/test_500_10000:
Serial: 10000:500:1728:0.048565:0.049043
OMP 2: 10000:500:1728:0.059204:0.059424
OMP 4: 10000:500:1728:0.062154:0.062277
OMP 8: 10000:500:1728:0.380521:0.380629
=====
File /home/tkt1/hpc-project/testbed/test_50_100:
Serial: 100:50:700:0.000129:0.000473
OMP 2: 100:50:700:0.000396:0.000451
OMP 4: 100:50:700:0.000595:0.000649
OMP 8: 100:50:700:0.002719:0.002772
=====
File /home/tkt1/hpc-project/testbed/test_50_1000:
Serial: 1000:50:533:0.000858:0.001118
OMP 2: 1000:50:533:0.001454:0.001509
OMP 4: 1000:50:533:0.001884:0.001936
OMP 8: 1000:50:533:0.005637:0.005692
=====
File /home/tkt1/hpc-project/testbed/test_50_10000:
Serial: 10000:50:546:0.008427:0.008699
OMP 2: 10000:50:546:0.011097:0.011154
OMP 4: 10000:50:546:0.006733:0.006762
OMP 8: 10000:50:546:0.024972:0.025030
=====
File /home/tkt1/hpc-project/testbed/test_5_5:
Serial: 5:5:129:0.000018:0.000265
OMP 2: 5:5:129:0.000163:0.000202
OMP 4: 5:5:129:0.000253:0.000291
OMP 8: 5:5:129:0.000610:0.000661
=====
"results" 133L, 4206C                                     118,1      Bot
```

Figura 7 - Results

Conclusions

We can observe that have three files that make failure with error of the threads and execution, we also can observe all the executions with 8 threads take longer time;

I conclude with the smaller inputs not have much difference in the serial and the parallel with 2 and 4 threads. A case "test_50000_1000" for serial was generated 3.9 sec and for the parallel 6.0 and 6.4, i suppose that some bottleneck in the execution, because the another result with the same quantity increasing the value of weight and results giving similar.

The another way we can test its transform the used matrix to vector because the indexing of vector its more faster, but the address the matrix in the memory its a similar of vector.

This graph we can observe that the unique that have a difference when the quantity of input increase its the 8 threads.

